AD710363

United States
Naval Postgraduate School

THESIS

A DICTIONARY STRUCTURE FOR USE WITH AN
ENGLISH LANGUAGE PREPROCESSOR TO A
COMPUTERIZED INFORMATION RETRIEVAL SYSTEM

by

Charles Thomas Schmidt

June 1970

20050405094

51

A Dictionary Structure for Use with an English Language Preprocessor to a Computerized Information Retrieval System

by

Charles Thomas Schmidt
Lieutenant Commander, United States Navy
B.A., University of Michigan, 1962

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the
NAVAL POSTGRADUATE SCHOOL
June 1970

Author: C Thomas Schmidt

Approved by: George E. Heidorn
Thesis Advisor

Brooks J. Lockhart
Chairman, Committee for Computer Science

R. F. Rinehart
Academic Dean

ABSTRACT

This paper describes the formation of a dictionary list structure which can be used by an English language translator to enable natural language man-machine conversation directed towards the retrieval of information from a data bank. The hierarchical structure of the letters in a word and the placement of word attributes in this structure is discussed.

A computer program, which accepts as input an English language sentence and processes this sentence in conjunction with the dictionary list structure to obtain the attributes of the individual words, is described. The incorporation of this dictionary structure into a complete natural language information retrieval system is also discussed.

TABLE OF CONTENTS

Preceding page blank

LIST OF FIGURES

LIST OF DEFINITIONS

1. Record (Logical Record) - A collection of related data elements treated as a unit. (E.g., the topic name and its numeric code word).

2. File - A collection of related records treated as a unit.

3. Information Retrieval System - A process developed to recover specific information from a data bank.

4. Key-Searches - Searches conducted to retrieve information in accordance with a specific data input parameter (or set of parameters) called a key.

5. Pointer - A numeric value used to specify an address in core

6. Null Pointer - A special pointer value that cannot relate to any address in storage (represented by NIL or λ).

7. Node - One or more consecutive bytes of core storage divided into parts called fields. (synonym -- cell)

8. Tree - A finite set of one or more nodes such that there is one specially designated node called the root of the tree and the remaining nodes are partitioned into disjoint sets of trees which are called subtrees of the original tree.

9. Binary Tree - A tree structure where each node has at most two subtrees. When there is only one subtree present, a distinction is made between whether it is a left subtree or a right subtree.

10. Hash Coding - A method of effecting random storage where a transformation is performed on a key (in this case the topic name) to produce an address in the table which can be used to locate a position for placing the key and its associated item.

Preceding page blank

I. INTRODUCTION

The Technical Reports Section at the Naval Postgraduate School Library currently offers its users access to those reports in its holdings by either of two methods. First, the user can search through the card index to obtain the report number of the item desired, or, second, he can conduct a computerized library search to obtain a bibliography of those reports in the holdings which pertain to the subject area of interest.

This library search is conducted by entering a numeric code word into an information retrieval system which will then use this code word as a key to search for and retrieve a bibliographical listing of those reports which pertain to the entered code word. Any number of code words can be combined, and the items on the bibliography retrieved for the user will have all of the code words within their list of keys. At present the Technical Reports Section has about 100,000 bibliographical items in its file, and these items can be accessed by over 10,000 different code words.

Although a library search is faster and easier for the user, especially when only a general topic area is desired, it does have a number of shortcomings. First, there is a nominal delay between the request for the search and the obtaining of the bibliography resulting from thatsearch. Second, the user does not know at the time of his request whether or not his topic area is too broad or too narrow to obtain a reasonable bibliography. Third, the user presently has to personally search through a book of topic areas to obtain the numberic code words required for the information retrieval system.

Preceding page blank

This report is the result of a project to develop a flexible and accessible information system for the specific objective of retrieving a bibliography of the reports held by the Technical Reports Section of the Naval Postgraduate School Library. The proposed system would be implemented on the school's IBM 360/67 computer and would utilize the time-sharing capability of that system.

The system as conceived would be similar to the REAL ENGLISH system at the University of Pennsylvania [1]. It would consist of an English language translation capability which would be used to pre-process the user's English language request, which has been entered at the time-sharing terminal, to determine the specific item(s) desired. Then the numeric code associated with each item would be retrieved from a table and entered into the existing information retrieval system. The number of items in the bibliographical file which met the criterion of the code word(s) would be printed out at the terminal, and the user could then specify whether or not there was a sufficient number of bibliographical items for his use. If the number of items were not sufficient or overly abundant, the user could re-initiate his request in either a broader topic area or a more restricted area to obtain the desired number of items in the bibliography. If the number of bibliographical items was sufficient, the user could specify that the bibliography was to be printed out at the terminal or at the offline printer at the computer installation. A generalized block diagram of the proposed system is shown in figure 1, page 11.

The remainder of this report has been written in two parts. The first part describes the formation of a dictionary structure required for the English language translation, the processing of the user's

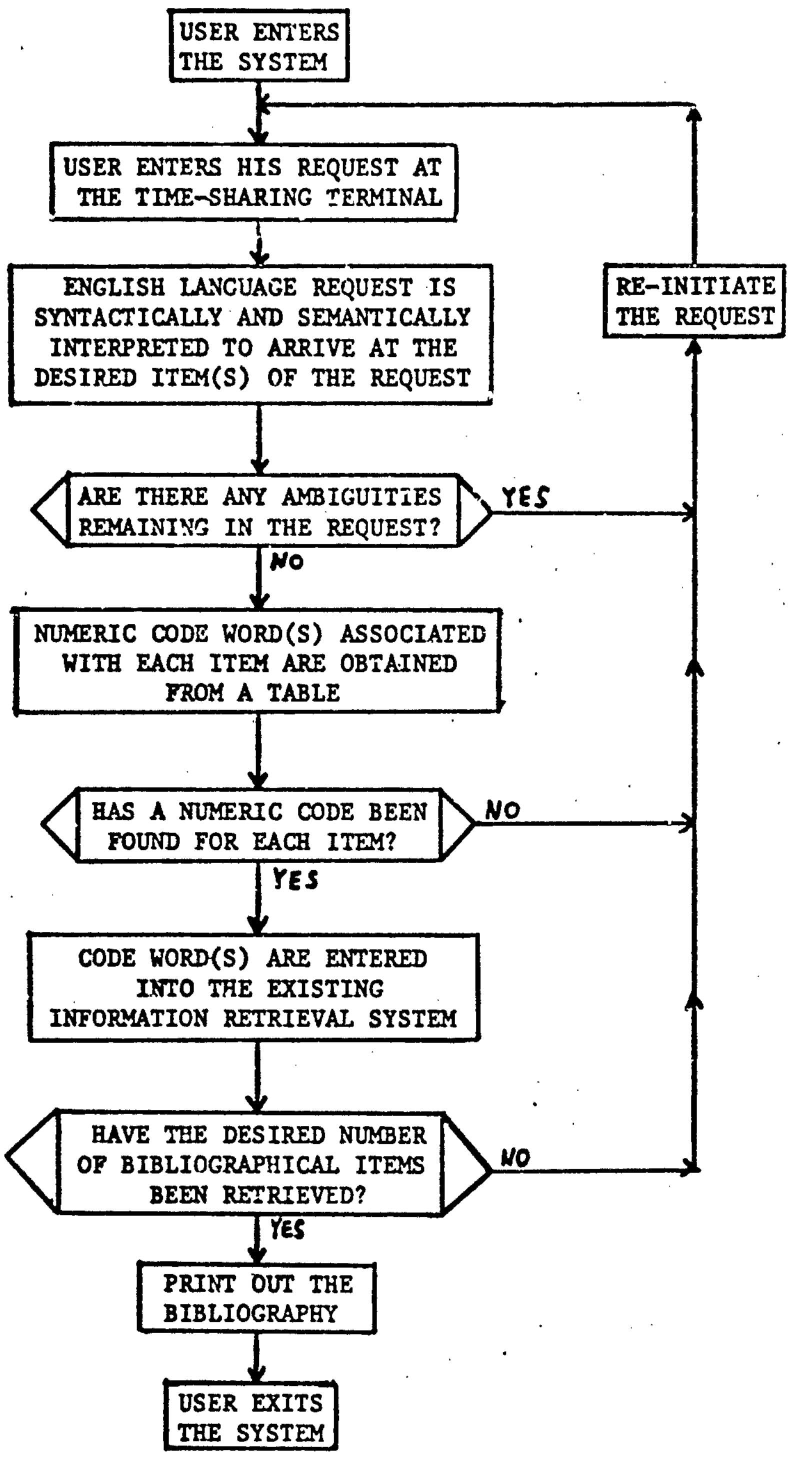

USER ENTERS THE SYSTEM
USER ENTERS HIS REQUEST AT THE TIME-SHARING TERMINAL
ENGLISH LANGUAGE REQUEST IS SYNTACTICALLY AND SEMANTICALLY INTERPRETED TO ARRIVE AT THE DESIRED ITEM(S) OF THE REQUEST
ARE THERE ANY AMBIGUITIES REMAINING IN THE REQUEST?
YES
NO
NUMERIC CODE WORD(S) ASSOCIATED WITH EACH ITEM ARE OBTAINED FROM A TABLE
HAS A NUMERIC CODE BEEN FOUND FOR EACH ITEM?
NO
YES
CODE WORD(S) ARE ENTERED INTO THE EXISTING INFORMATION RETRIEVAL SYSTEM
HAVE THE DESIRED NUMBER OF BIBLIOGRAPHICAL ITEMS BEEN RETRIEVED?
NO
YES
PRINT OUT THE BIBLIOGRAPHY
USER EXITS THE SYSTEM
RE-INITIATE THE REQUEST

FIGURE 1. System Block Diagram

request to obtain from the dictionary the attributes of the individual words, and the retrieval of the numeric codes from the table of code words. The second part consists of illustrative examples of the system.

The syntactic and semantic interpretation of the attributes of the request and the actual implementation with the existing information retrieval system are left for further study. Some work in the area of syntactic and semantic interpretation has been done by Cautin [1,2] and Kellog [5,6] which might be of value for anyone who desires to conduct further work on this system.

II. SYSTEM DESIGN

At this stage in the development of the system, there are three main computer programs or subroutines. The primary program, LIB-SER, which is mainly concerned with performing operations on lists, was written in PL/I [7,8,9] to take advantage of the list processing and recursive capabilities of that language. Also, PL/I was chosen instead of a list processing language, such as LISP 1.5, because of the availability of PL/I at the Naval Postgraduate School computer installation.

The TABLE program, which is also in PL/I, was written exclusively for use as a calling program for the construction and maintenance of the numeric codes table. PL/I was chosen for this routine mainly because of its character manipulation capabilities.

The final program, HASHX, is a subroutine written in IBM 360/67 Assembly language [10,11] and is called into operation by both the LIB-SER and TABLE programs. Assembly language was chosen since it allows a closer control over the representation and location of data while in computer core storage. This close control is necessary since HASHX is used to move the core character representations of the topic names and their numeric codes from one position to another in core storage.

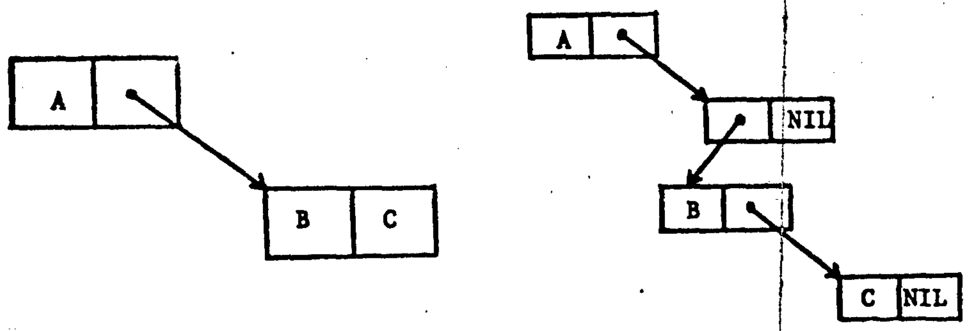
A. LIB-SER PROGRAM

The LIB-SER program is the main program utilized by the proposed library search system. It is utilized to build lists, add items to lists, delete items from lists, and search for items on the lists maintained during the operation of the proposed system. The lists of prime importance are ATOMLIST, the dictionary list (L1), the user's request list (L2), and the multiple definitions list (L3).

1. List Structure in the Program

A list structure was used in this program since a hierarchical structure was desired for the dictionary list. This hierarchical structure is used to simplify the search through the dictionary for the attributes of the words in the user's request. The inputs to the program, the dictionary file and the user's request, are built into a system of connected lists where these lists take on a form similar to the program and data structure utilized in the list processing language LISP 1.5 [12].

In LISP 1.5, program and data are represented in the form of symbolic expressions or S-expressions where these S-expressions are of indefinite length and have a branching binary tree structure which allows the identification and isolation of sub-expressions or sub-trees within the basic structure. The S-expressions are graphically represented in this binary tree structure, as shown in figure 2 on page 15, as a sequency of nodes or cells, each of which has two fields. When using the dotted pair notation in LISP 1.5, both of the fields can be used to contain a pointer to another node or cell or they can be used to contain a character string. In an alternative notation, list notation, the left field is the only field which can be used to contain a character string or pointer. The right field contains a pointer to the next node in the tree or a special pointer, NIL, which is used as a terminator of the list.



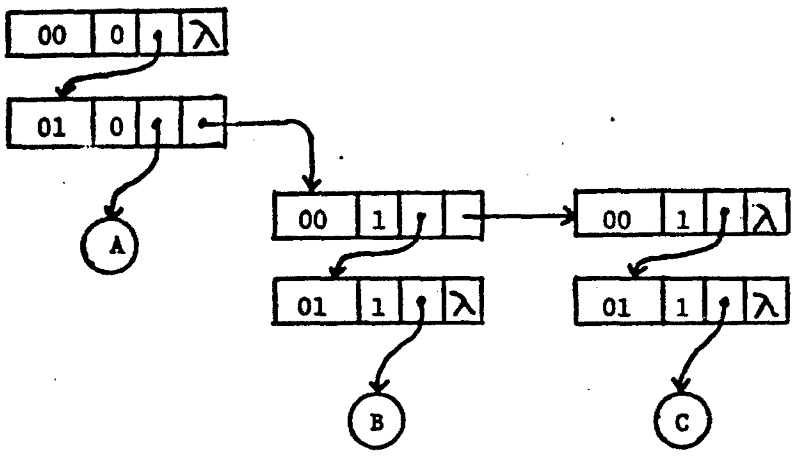
(A·(B·C)) (A(B C))

(a) dotted pair notation (b) list notation

FIGURE 2. Graphical Structure of LISP 1.5 Lists

The list structure used in this program is a modified version of the list notation used in LISP 1.5. This modification was required by the use of PL/I as the programming language and the requirement of this language that the contents of the fields in a cell be explicitly declared. Thus, no one field can be used to contain both a pointer and a character string during a program execution. This required that the character strings be placed in a type of cell, called ATOMCELL in this program, defined separately from the connective cells, called CONCELL, used to represent the tree structure. It also required, as shown in figure 3 on page 16, that each of the character strings which is placed in an ATOMCELL to have associated with it two CONCELLS. The upper CONCELL is used to connect the character string to other character strings at the same level in the subtree. The lower CONCELL is used to contain a pointer to the character string contained in the ATOMCELL and serve as an attachment point for any subtrees of that character string.

Since there are two CONCELLS associated with each character string, a method of determining whether or not the left field of the CONCELL pointed to an ATOMCELL or to another CONCELL in the list was required. This was accomplished by the addition of a third field to the CONCELL type of cell. A fourth field was added to indicate the depth of the occurrence of the cell in the list structure and is utilized during the construction of the trees.



(A<BC>)

FIGURE 3. Graphical Structure of the Lists Used.[1]

a. ATOMLIST List

The ATOMLIST list is used to store the alphabetic, numeric, and special characters which are required by the program to be retained in their symbolic form during the operation of the program. It consists of a series of connected cells called ATOMCELLS, shown in figure 4 on page 17, which have three fields. Each new cell added to the list is connected to the first cell on the list.

[1]All of the circled character strings are included in the ATOMLIST list and only conceptually displayed here. The λ or X used in the CDRF field is used to indicate the NULL pointer.

The three fields are the SIZE field, which contains the number of characters of the item represented in the cell, the PNAME field, which contains the character representation of the item, and the NEXT field, which contains a pointer to the next ATOMCELL on the list.

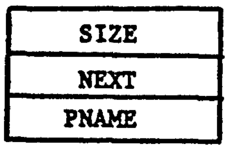


FIGURE 4. Structure of an ATOMCELL

Each of these ATOMCELLs is placed on this list as the characters they contain are encountered during the operation of the program, with the majority of them being attached to the list during the initialization phase of the dictionary. Each item is included in this list only once since a search of the list (procedure LOOKUP) is conducted every time an item is encountered to determine if an ATOMCELL has or has not already been included in the list which contains the character(s) of the item. No ATOMCELLs are deleted from this list during the operation of the program.

All referrals to the character representation of any item on the ATOMLIST is by the pointer to the entire ATOMCELL. This pointer will be the same as that contained in the NEXT field of the ATOMCELL immediately prior to the ATOMCELL containing the desired characters, if one exists, or else it will be the pointer ATOMLIST since the characters would then have to be contained in the first ATOMCELL on the list. A diagram of the ATOMLIST is shown below in figure 5.

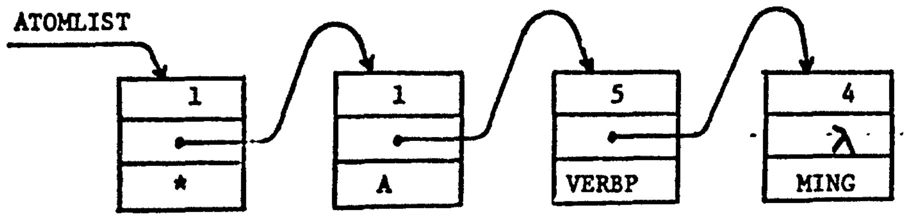


FIGURE 5. Diagram of the ATOMLIST

b. Connective Lists

All of the remaining lists built and used by the program are used to connect the character representations in the ATOMLIST in various fashions. These lists consist of a series of CONCELLS which have four fields, an ATOMB field, a LEVEL field, and two pointer fields, CARF and CDRF. The structure of the CONCELL is shown in figure 6.

ATOMB	LEVEL	CARF	CDRF

FIGURE 6. Structure of a CONCELL

The ATOMB field is used to indicate the purpose of the CONCELL. It is a two bit field which can contain one of three binary values, 00, 01, or 11. The 00 value indicates that the CONCELL is being used as a connective cell where both the CARF and the CDRF fields contain pointers to other CONCELLs, with the exception that the CDRF field may contain a null pointer. All items in the same subtree at any particular level will be connected by the CDRF field of a CONCELL with a 00 ATOMB field.

The 01 value indicates that the CARF field contains a pointer to an ATOMCELL. The CDRF field may contain a pointer to another CONCELL with a 00 ATOMB field which is at the next level down in the tree, or else contain a null pointer. The 11 value, which can only be placed in the ATOMB field during the initialization phase of the dictionary list, indicates that the CARF field contains a pointer to an ATOMCELL which contains the name of a multiple definition or an attribute name. The CDRF field always contains a null pointer. A sample connective list is shown below in figure 7.

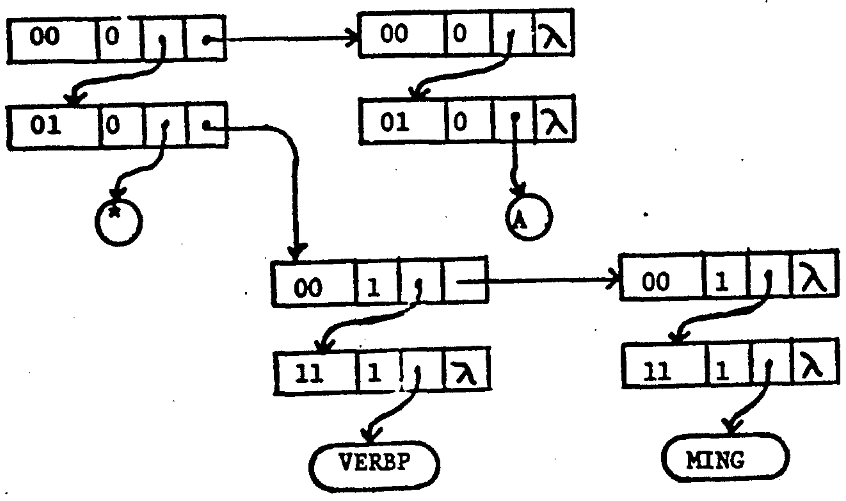


FIGURE 7. A Connected List

The LEVEL field contains a numeric indication as to the level in the tree structure of the particular cell. For CONCELLs that are contained in the multiple definitions list, the value contained in the LEVEL field is not changed when that definition is included at other places in the structure of the dictionary.

The use of the multiple definitions will be explained later during the discussion of the initialization of the dictionary.

2. File Descriptions

Two files are used by the program during the initialization phase to enter data into the program, the Dictionary file and the Codes file. Both files are read-only files when used by the LIB-SER program.

a. Description of the Dictionary File

(1) Convention and Special Characters. With the exception of the attribute names and the names of the multiple definitions sequences, all items in the dictionary are single characters. Spaces or blanks are used only to improve the readability or to separate items in the dictionary. The attribute names and the multiple definition names must be at least two characters in length and the multiple definition names start with the letter M.

There are eight characters used in this file which have special significance during the operation of the program. These characters are the right and left parentheses, the greater-than symbol (>), the less-than symbol (<), the dollar sign ($), the minus sign (-), the asterisk (*), and the slash(/).

The six characters, right and left parentheses, greater-than symbol, less-than symbol, the dollar sign, and the slash, are used during the building of the list structure to determine the proper connections which have to be made. The left parenthesis is used to indicate the beginning of the dictionary file. The right parenthesis is used to indicate the end of the dictionary file. The dollar sign is used to terminate the processing of the current input. The less-than symbol is used to indicate that the items that follow are to be attached to the preceding character at the next lower level in the tree structure. The greater-than symbol is used to indicate that items at the current level in the subtree have been processed and that the following items are not to be attached at that level but at the next higher in the tree structure. The slash is used to indicate that the next item is the name of a multiple definition and that the previously formed subtree, which is attached to that multiple definition name, is to be inserted at this point in the current subtree.

The two characters, the minus sign and the asterisk, are used during the attribute determination phase to indicate special conditions or operations. The minus sign is used to indicate that the following attribute is not to be retained for the word thus formed. This attribute may or may not have been retrieved for the word under consideration. The asterisk is used to indicate that a complete word has been placed in the tree structure to that point in the tree. It is also used as the attachment point for any remaining attributes of the word that it terminates.

(2) File Description. The Dictionary File consists of a sequence of characters contained in a series of sixty-character records. The file begins with the left parenthesis and ends with a right parenthesis and a dollar sign in the last record. The first character in the sequence after the left parenthesis is the symbol one (1) which is used as an attachment point for all of the multiple definitions. The remaining characters are placed into the file such that the tree structure which is built from this file will contain all of the words entered into the dictionary of the program.

At the zero level in the dictionary tree, the twenty-six letters of the Latin alphabet are attached to the preceding character in descending order according to their probability of occurrance in the English language [13]. These letters are used as the first character of all words in the dictionary. Each of the remaining letters in a word is preceded by the less-than symbol to indicate that they are to be attached to the previous letter in the word at the next lower level in the subtree.

Where there are attributes that are common to a number of words with similar beginnings, those attributes can be included in the sequence immediately prior to the beginning of the dissimilarity. Wherever a complete word has been represented, an asterisk is included in the sequence following the last letter of the word. When a word being described has a common ending, such as the plural endings ES and S, the multiple definition name for that ending is placed in the sequence immediately following the slash symbol, and this multiple definition name is used to complete the word. Figure 8 illustrates the use of the multiple definition name in the sequence.

```
S<O<M<E<* ADVBP>>>>   E<N<D <VERBP * /MING>>>>
```

FIGURE 8. Sample Dictionary Record

b. Codes File.

The codes file consists of a series of sixty-character records of which only the first twenty characters are utilized. Each record consists of the topic name beginning in the first character position and followed by the five number numeric code word in the last five character positions with blanks in between. The records are arranged in the file according to the hash coded index of each item that is determined during the operation of the TABLE program and its call to the subroutine HASHX .

3. Initialization.

The program has an initialization phase in which the dictionary file is read and structured into the tree and subtrees required for later use in the determination of the attributes of a user's request. After the dictonary has been structured, the table file is read into an array for use during the search for the numeric code words required for the information retrieval system.

a. The Dictionary Tree Structure.

The dictionary tree structure is a connective list which has pointers in the CARF field of the appropriate CONCELL to point to the required symbolic data in the ATOMLIST. It is formed by first constructing the multiple definitions subtree and then constructing the remaining subtrees, all of which start with the letters of the alphabet and some punctuation characters. All of the subtrees are constructed independently by the recursiveness of the structuring procedure (S_EXPR) until the last subtree has been structured at which time the subtrees are connected to form the dictionary tree. The recursive construction of the tree is illustrated in figure 9 on page 24.

Although the multiple definitions subtree is constructed in the same manner as all of the other subtrees in the dictionary, there is a special pointer (L3) which is assigned to point to this subtree during the initialization. This assignment occurs immediately upon completion of the structuring of the subtree and before the next subtree is started. This pointer is required during the structuring of the remaining subtrees as a beginning pointer for the procedure (LOOKLIST) when a search for a multiple definition used in the other subtrees is required. The multiple definitions subtree cannot be recursively defined (i.e., a multiple definition which uses another multiple definition as one of its endings) since the special pointer is not assigned until the completion of the entire structuring of the subtree.

During the construction of the remaining subtrees, the multiple definitions are used whenever a slash is encountered during

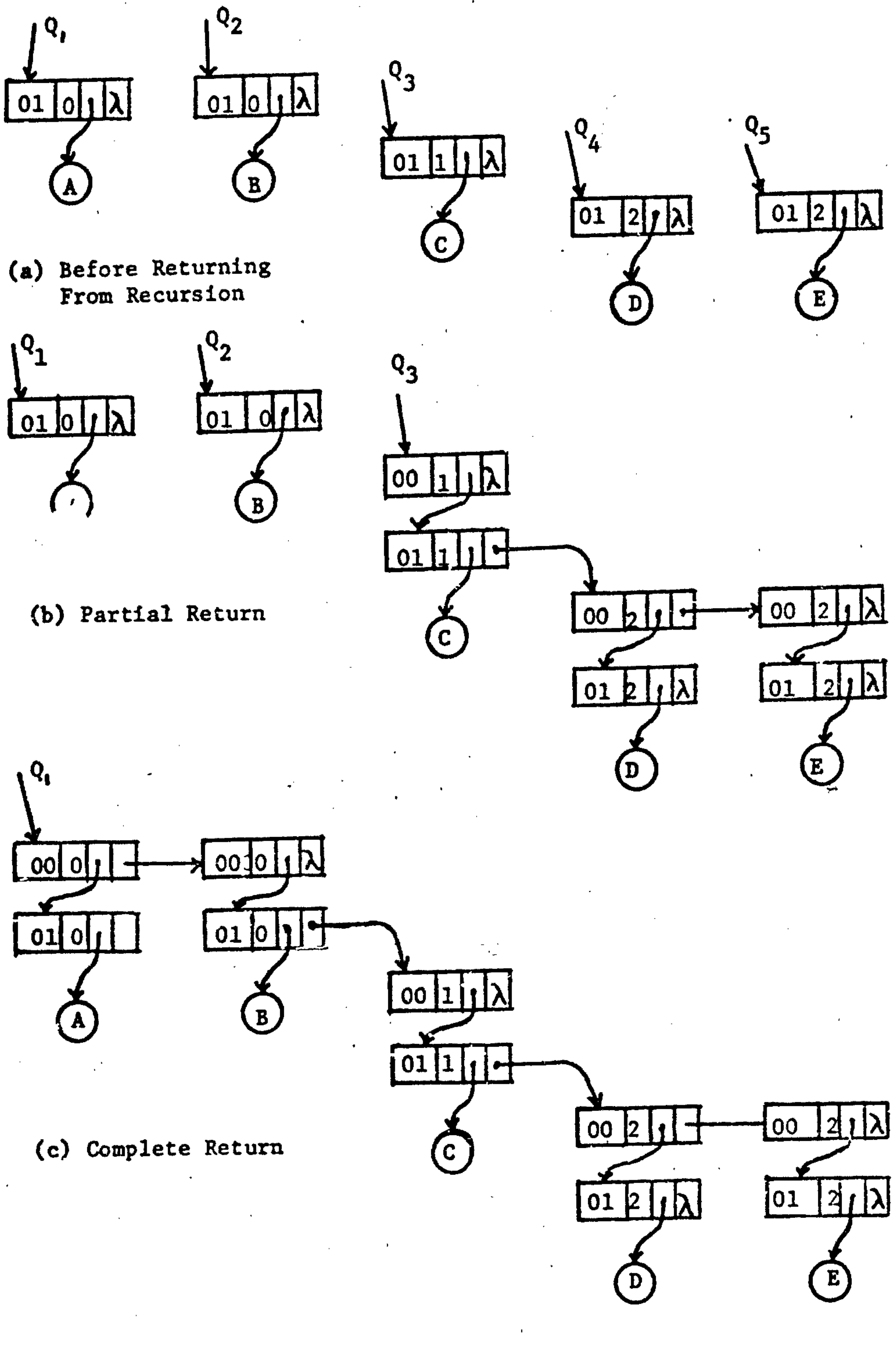


FIGURE 9. Recursive Structure of Connected Lists

the scanning of the input symbols. When this occurs, the procedure LOOKLIST is invoked to determine the pointer to the subtree required at this point for the completion of the word being entered into the dictionary. The program allows more than one multiple definition to be used as an ending of a word with the stipulations that none of the multiple definition subtrees can begin with the same letter nor can the multiple definitions subtrees begin with the same letter as is already included in the subtree of the word at that level. The use of the multiple definition is illustrated in figure 10 on pages 26 and 27.

b. The Table Array

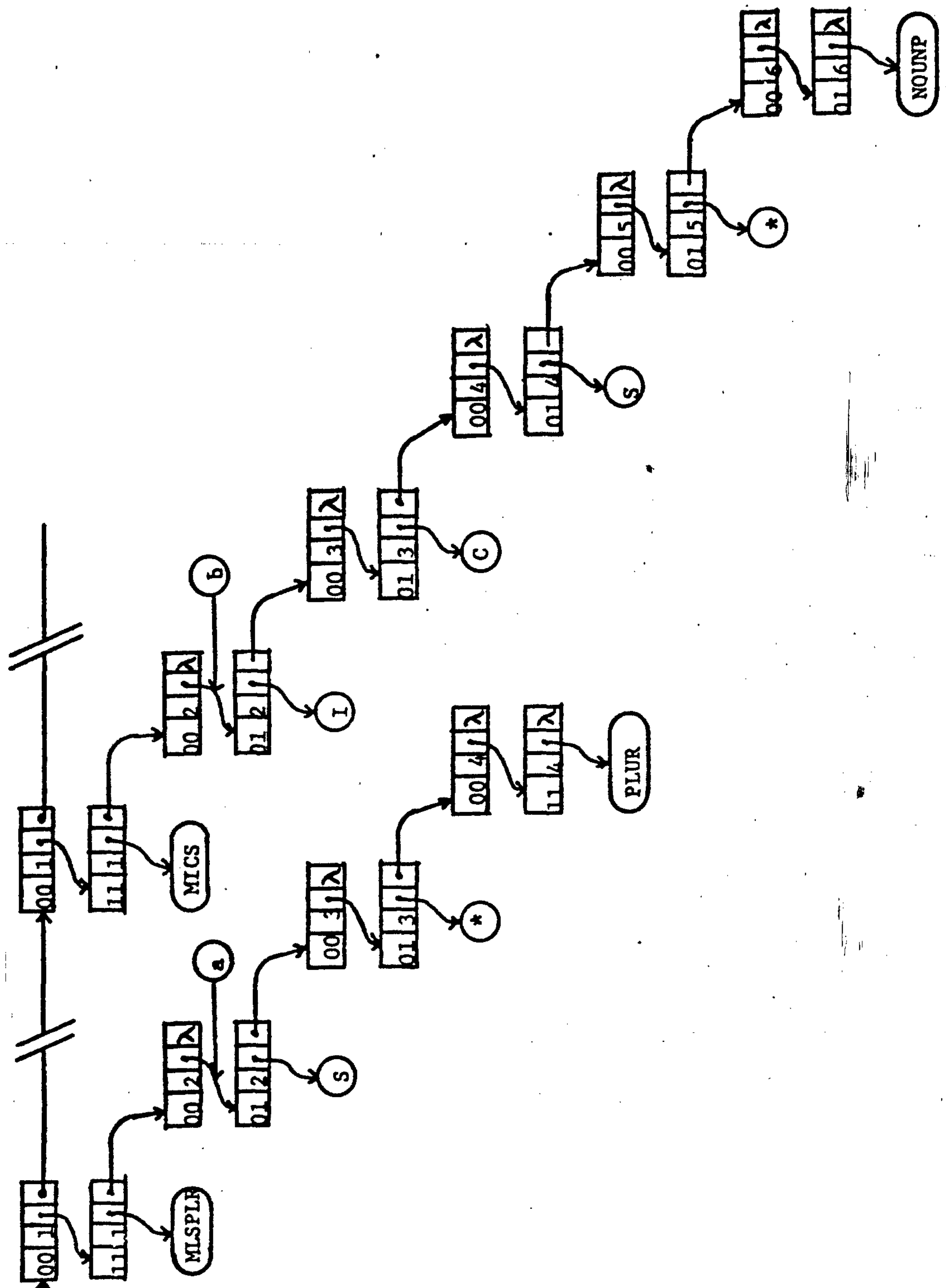
The table file is copied directly into the array area, CODE. No processing occurs with the items in the file during the initialization phase of the program. Currently, there is a logical switch (true or false) which has been turned off (false) to prevent the table file from being read into the CODE array. When further implementation of the proposed library search system has been accomplished, this switch can be turned on to allow the initialization of the table array for use in the table search procedure.

4. Processing of a Request

After the initialization phase has been completed, the computer will loop through the remaining parts of the program until the string STOP$ is entered at the terminal by the operator.

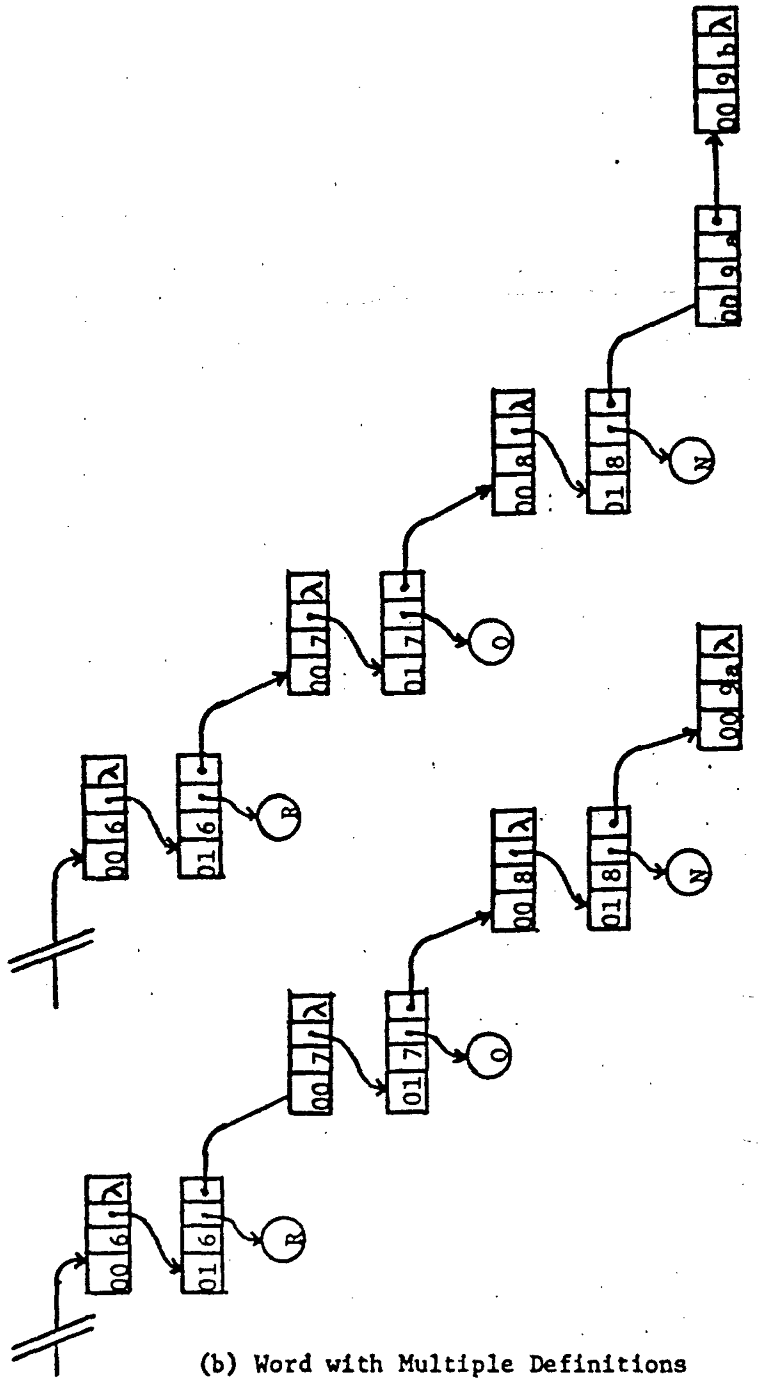
a. Entry of a Request

The user is requested by the program to enter his name and student box number (or other appropriate school mail box) to be used for identification purposes. The user is then asked to enter



(a) The Multiple Definitions

FIGURE 10. Attachment of Multiple Definitions



(b) Word with Multiple Definitions

FIGURE 10. Attachment of Multiple Definitions

his English language request surrounded by right and left parentheses and ending with a dollar sign. If the request is too long to be entered at one time, the right parenthesis and dollar sign can be left off until the entire request has been entered.

The request entered will be built into a tree (L2) in the same manner as the dictionary tree where all of the words and punctuation are entered as single item. Figure 11 is an example of a typical user's request tree.

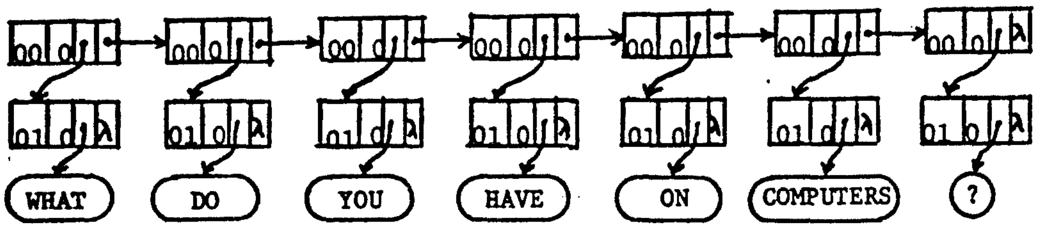


FIGURE 11. Sample User's Request

This tree structure will then be passed to the attribute determination routine for processing.

b. Determination of the Attributes

In the procedure ATTRB, each word in the user's request is separated into its constitutent letters and these letters are passed one at a time to the LOOKLIST procedure. The ATTRB procedure will also pass to the LOOKLIST procedure the subtree, or tree if the letter is the first letter of the word, on which the LOOKLIST procedure is to search for the letter.

In the LOOKLIST procedure, each letter is searched for only in the highest level of the subtree passed to it. Any attributes which are encountered before the letter is found will be attached to a temporary pointer (L3) for later combination with the final attributes.

When the letter has been found, the pointer contained in the CDRF field of the CONCELL which points to the ATOMCELL containing the letter is returned to the ATTRB procedure. This pointer will be used by the ATTRB procedure as the pointer to the subtree used during the next call to LOOKLIST.

This searching will continue until all of the letters of the word have been found. After the last letter has been processed, the ATTRB procedure will pass to the LOOKLIST procedure an asterisk and the latest subtree. The pointer (Q) returned from this call will point to the list of final attributes of the word, if any.

At this point, both the temporary attribute list and the final attribute list will be passed to the DELETE procedure for the deletion of any attributes in the temporary list which are indicated as not applying to the word by the prefixed minus sign on the attribute. The before and after structure of the temporary and final attribute lists are illustrated in figure 12.

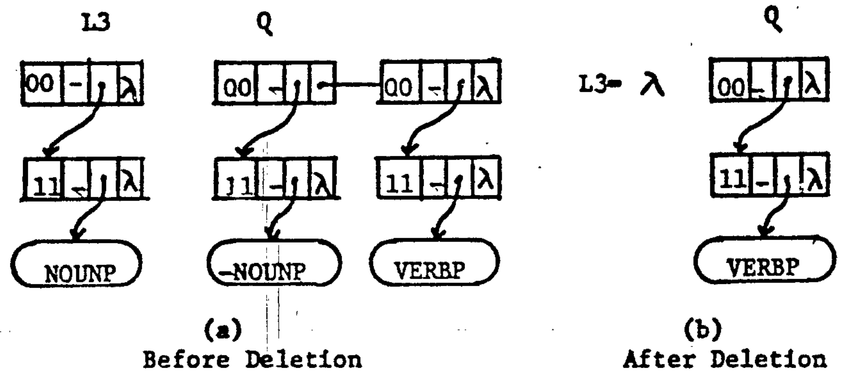


FIGURE 12. Temporary and Final Attribute Lists

The temporary list and the final list will be connected after this deletion has taken place, and the resulting list will be attached to the word on the user's request list (L2) as shown in figure 13.

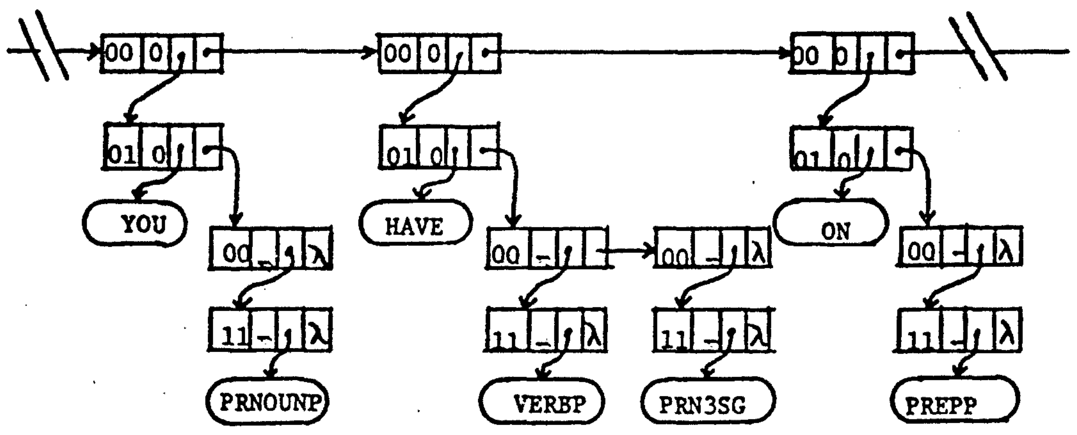


FIGURE 13. User's Request with Attributes Attached

As the processing of each word has been accomplished, a message is displayed at the terminal to the effect that the attributes have been found. If during the search for the letter of a word by the LOOKLIST procedure, that letter is not found, a null pointer is returned to the ATTRB procedure. This null pointer will result in the ATTRB procedure displaying a message at the terminal that the attributes of the word have not been found, and then the procedure will begin to process the next word in the request.

After all of the words have been processed, the computer prints out the user's list which now includes the attributes of the individual words. The computer then branches back to the beginning and requests a new user to enter his name and box number.

B. TABLE PROGRAM

Since the TABLE procedure in the main program, LIB-SER, was written to allow only the retrieval of the numeric code words from the CODE array, the auxiliary program, TABLE, was written. The TABLE program allows the entry of an item, the deletion of an item, and the retrieval of information contained in an item in the CODE array.

The TABLE program first initializes the CODE array by copying directly into the array the individual records in the CODES file. The records are truncated after the twentieth character. Once in the array, the program calls the subroutine HASHX with various parameters to perform the functions of retrieval, deletion, or storage of an item in the array. When the required modifications to the array have been accomplished, the CODE array is copies back into the CODES file with each item being padded with blanks on the right to fill out each record.

Although the TABLE program was written to be operated from the time-sharing terminal, with slight modifications it could be run under the batch processor at the computer installation. This might be desirable if the number of modifications is substantial.

C. HASHX SUBROUTINE

The HASHX subroutine, which is used to compute the address of a location in a table or array and either return the contents of that location or place an item in that location, is based on the work done in scatter storage techniques by W. D. MAURER [14]. The method used here is referred to as a "division hash code" to distinguish it from the logical or multiplicative methods most often used in scatter storage systems [15].

Hash coding techniques can be applied to any table or array in which the access is to be made to the entries in an unpredictable order and the items are identified by some key or name associated with their contents. In the hash coding technique, a transformation is performed on the key to produce an address in the table where the key and the entry associated with that key can be placed. A good transformation is one which will distribute these addresses uniformly across the available table area.

In most hash coding techniques, the table size is restricted to being a power of two, since the common method of obtaining the address is to calculate a k-bit field which is assumed to be a random integer between zero and 2^k-1, and this integer is then used as the address. In the method proposed by MAURER and used in this subroutine, the table size is a prime number, and therefore it can be almost any size desired.

1. Determination of the Hash Code

The hash coding method used here consists of using the first eight characters of the topic name as the key. The first four characters are exclusively OR-ed with the last four characters to obtain one full computer word (32 bits on the IBM 360/67 computer), and this computer word is divided, using integer arithmetic, by the table size. The remainder from this division, which is a number between zero and one-less-than-the-table-size, is used as the hash code.

This method of calculating the hash code is comparable in speed to the multiplicative and logical methods with the advantage that is almost completely free of the nonrandomness which often occurs in the other methods.

In using this method of hash coding, a change in the table size will change all of the hash codes for the table entries. If the table size has to be changed, all of the hash codes for the entries must be recomputed. By using a prime number which is twenty-five percent larger than the maximum number of expected entries for the table, the average number of steps required to find an item in the table is less than two, i.e., the original hash code address and one collision modification.

2. Collision Handling

In this method, as in all other hash coding methods, when two keys have the same hash code, a "collision" is said to have occurred. When a collision occurs, the item causing the collision must be located out of place in the table.

There are a number of methods for determining a location for this item, such as searching linearly forward in the table. Another approach is to add a random number to the collision hash code to determine a new location. Neither of these methods is very satisfactory since they are intrinsically slow if there is any sort of clustering of the items in the table (many items with the same hash code place next to each other).

The method of handling collisions used in this subroutine is the "quadratic search". In this method, a quadratic equation is used to calculate the new location. The hash code which caused the collision is used as the constant term in the equation and the other coefficients depend upon the table size. The quadratic search was shown by MAURER to always search for the item or an empty location by looking at exactly half of the remaining table locations if the table size is a prime number. In using this method of handling collisions, the table is declared full when the quadratic search has searched through half of the table.

III. AN EXAMPLE

To demonstrate the operation of the LIB-SER program, only a representative dictionary is utilized. For a total implementation of this system, the dictionary would have to be greatly expanded. The dictionary used for this demonstration contains seventy words.

The program is loaded for operation at the time-sharing terminal by entering LDRM. LDRM is a time-sharing executive file which causes the program and the system library to be loaded into the computer and starts the execution of the program. If a non-blank character had been entered after LDRM, the program would request the operator to enter optional parameters for use by the program. These parameters are logical switches which can be used to display traces through the program or selectively enable/disable the various routines in the program.

After execution has begun, and any optional parameters entered, the program initializes the dictionary tree structure. The computer will then ask a user to enter his request, which will then be processed to obtain the attributes of the individual words. Figure 15 illustrates a complete run for an individual user.

The difference between the attributes for a word which has been processed with the DELETE procedure disabled and the same word when it has been enabled is illustrated below in figure 14.

The computer will then loop back to ask a new user to enter his request. The program can be stopped by entering STOP$.

```
(INFORM⟨NOUNP -NOUNP VERBP⟩)          (INFORM⟨VERBP⟩)
```

(a)	(b)
DELTE DISABLED	DELETE ENABLED

FIGURE 14. Effects of DELETE Routine

```
ENTER NAME AND SMC NUMBER
_c.t. schmidt box 1962

ENTER REQUEST, SURROUND WITH ( )$
_(what do you have on computers?)$
WHAT
DO
YOU
HAVE
ON
COMPUTERS
?

       *******************************
          C.T. SCHMIDT BOX 1962
       *******************************


ATTRIBUTES FOUND FOR   WHAT
ATTRIBUTES FOUND FOR   DO
ATTRIBUTES FOUND FOR   YOU
ATTRIBUTES FOUND FOR   HAVE
ATTRIBUTES FOUND FOR   ON
ATTRIBUTES FOUND FOR   COMPUTERS
ATTRIBUTES FOUND FOR   ?

(WHAT<INTER ADVBP> DO<VERBP>  YOU<PRNOUNP>  HAVE<VERBP PRN3SG>
 ON<PREPP>  COMPUTERS<NOUNP PLUR>  ?<PUNTI>)

** AT THIS POINT YOUR REQUEST WOULD BE PASSED TO THE
   TRANSLATION AND TABLE SEARCH ROUTINES. **
```

FIGURE 15. Sample Program

IV. CONCLUDING REMARKS

This paper has discussed a dictionary list structure which could be used by an English Language translator to enable natural language man-machine conversation directed towards the retrieval of information from a data bank. The development of the English Language translator utilizing this dictionary structure and the inclusion of this translator into the existing information retrieval system used by the Technical Reports Section at the Naval Postgraduate School Library would provide the school a flexible and accessible information system for use in obtaining a bibliography of the reports held by the library.

A translation procedure which is based on a generative model of syntax and semantics that is comprehensive enough to automatically resolve some forms of syntactic and semantic ambiguities would be required for this system. The translator would have to be able to syntactically parse the user's request, using the attributes determined from a dictionary structure such as described in this paper, and then use this parse in conjunction with the semantic environment of the system to determine the topic areas of interest from the user's request. In the accomplishment of this goal, it should have some means of communication with the user to resolve any ambiguities which it is not otherwise able to resolve. Once the topic areas of interest have been determined, the system can readily produce the numeric code words required to obtain a bibliography from the existing information retrieval system.

```
       /*****************************************************
       ******************************************************
       **                                                  **
       **              LIBRARY SEARCH SYSTEM               **
       **                                                  **
       **                 LIB_SER PROGRAM                  **
       **                                                  **
       ******************************************************
       *****************************************************/

LIB_SER: PROC(PARMS) OPTIONS(MAIN):
   DCL CODES FILE STREAM ENVIRONMENT (F(80)),
       DICT FILE STREAM ENVIRONMENT (F(80)):
   DCL PARMS CHAR(8)VAR,CODE(11) CHAR(20):
   DCL (ATOMLIST,P,NIL)PTR,CSIZE FIXED BIN,

      /*****************************************************
      *    DECLARATION OF THE FIELDS OF THE CELLS ON       *
      *    THE 'ATOMLIST' LIST.                            *
      *****************************************************/

   1 ATOMCELL BASED(P),
     2 SIZE FIXED BIN,
     2 NEXT PTR,
     2 PNAME CHAR(CSIZE REFER(SIZE)),

      /*****************************************************
      *    DECLARATION OF THE FIELDS OF THE CELLS ON       *
      *    L1, L2, AND L3 LISTS.                           *
      *****************************************************/

   1 CONCELL BASED(P),
     2 ATOMB BIT(2),
     2 LEVEL FIXED BIN,
     2 CARF PTR,
     2 CDRF PTR:

      /*****************************************************
      *    DECLARATION OF A FILE CONTROL BLOCK USED        *
      *    DURING THE READING AND WRITING OF THE           *
      *    FILES USED.                                     *
      *****************************************************/

   DCL 1 FCB STATIC,
       2 COMMAND CHAR(8),
       2 FILENAME CHAR(8) INIT('DICT'),
       2 FILETYPE CHAR(8) INIT('DATA'),
       2 CARDNUM FIXED BIN,
       2 STATUS FIXED BIN INIT(0),
       2 CARD_BUFF CHAR(80):

      /*****************************************************
      *    PROCEDURE WHICH IS USED TO INITIALIZE THE       *
      *    DICTIONARY AND THE TABLE ARRAY.                 *
      *****************************************************/

   DCL INIT ENTRY:
   INIT: PROC:
       DCL I FIXED BIN:
       ATOMLIST,NIL=NULL:
       FLD = '1'B:
       COMMAND = 'RDBUF':
       DISPLAY('INITIALIZING FROM FILE'):
       L1 = READS:
       COMMAND = 'FINIS':
       CALL IHEFILE(FCB):
       FLD = '0'B:
```

```
   IF TRI THEN CALL PRINTS(CDR(L1));
   IF TBL THEN DO;
     COMMAND = 'RDBUF';
     FILENAME = 'CODES';
        DO I = 1 TO TBLSIZE;
        CARDNUM = I;
        CALL IHEFILE(FCB);
        CODE(I) = CARD_BUFF;
        IF TRI THEN DISPLAY('CODE('||I||')  '||
          CODE(I)); END;
     COMMAND = 'FINIS';
     CALL IHEFILE(FCB); END;
  DISPLAY('END OF INITIALIZATION');
  END INIT;

 /***********************************************************
 *    PROCEDURE WHICH IS USED TO ENTER ITEMS               *
 *    INTO THE PROGRAM.                                    *
 ***********************************************************/

DCL INPUT ENTRY(CHAR(72));
INPUT: PROC(B);
  DCL B CHAR(72),(S CHAR(8),I FIXED BIN INIT(0))STATIC;
  IF FLD THEN DO;
    I = I+1;
    CARDNUM = I;
    CALL IHEFILE(FCB);
    B = CARD_BUFF;
    IF TRI THEN DISPLAY('BUFFER  '||B); END;
  ELSE DISPLAY('ENTER STRING, SURROUND WITH ( )$')
    REPLY(B);
  END INPUT;

 /***********************************************************
 *    PROCEDURE WHICH IS USED TO PRINT OUT ITEMS           *
 *  FROM THE PROGRAM.                                      *
 ***********************************************************/

DCL OUTPUT ENTRY(CHAR(*));
OUTPUT: PROC(C);
  DCL C CHAR(*),I FIXED BIN INIT(0)STATIC;
  IF FLD THEN DO;
    I = I+1;
    CARDNUM = I;
    COMMAND = 'WRBUF';
    CALL IHEFILE(FCB); END;
  ELSE DISPLAY(C);
  END OUTPUT;

 /***********************************************************
 *    PROCEDURE USED TO DETERMINE THE FUNCTION             *
 *    OF THE CELL SPECIFIED BY THE POINTER.                *
 ***********************************************************/

DCL ATOM ENTRY(PTR) RETURNS(BIT(2));
ATOM: PROC(A) BIT(2);
  IF A=NIL THEN RETURN('01'B);
  RETURN(A->ATOMB);
  END ATOM;

 /***********************************************************
 *    PROCEDURE WHICH CONSTRUCTS A CHARACTER               *
 *    POINTER CELL (A CELL WITH EITHER A 11 OR 01          *
 *    ATOMB FIELD) AND INSERTS THE POINTER                 *
 *    TO THE CHARACTER(S) ON THE 'ATOMLIST' LIST           *
 *    IF ALREADY ON THE LIST OR ELSE PLACES                *
 *    THE CHARACTER(S) ON THE 'ATOMLIST' LIST              *
 *    AND INSERTS THE POINTER TO THIS NEW CELL.            *
 ***********************************************************/

DCL STRING ENTRY(CHAR(*)VAR) RETURNS(PTR);
STRING: PROC(C) PTR;
```

```
          DCL C CHAR(*)VAR,(Q PTR,L FIXED BIN)STATIC;
          IF TRC THEN DISPLAY('STRING '||C||' AT LEVEL'||LEV);
          Q = LOOKUP(C);
          ALLOCATE CONCELL SET(P);
          L = LENGTH(C);
          IF L>1 THEN
             IF FLD THEN ATOMB = '11'B;
             ELSE ATOMB = '01'B;
          ELSE ATOMB = '01'B;
          CDRF = NIL;
          LEVEL = LEV;
          IF Q = NIL THEN DO;
             CSIZE = L;
             ALLOCATE ATOMCELL SET(Q);
             Q->NEXT = ATOMLIST;
             Q->PNAME = C;
             ATOMLIST = Q; END;
          CARF = Q;
          RETURN(P);
          END STRING;

        /*************************************************
        *    PROCEDURE WHICH SEARCHES THROUGH THE        *
        *    'ATOMLIST' LIST TO FIND THE CELL LOCATION   *
        *    OF THE INPUT CHARACTER(S).                  *
        *************************************************/

       DCL LOOKUP ENTRY(CHAR(*)VAR) RETURNS(PTR);
       LOOKUP: PROC(C) PTR;
          DCL C CHAR(*)VAR,Q PTR STATIC;
          Q = ATOMLIST;
             DO WHILE(Q¬=NIL);
             IF Q->PNAME = C THEN RETURN(Q);
             Q = Q->NEXT; END;
          RETURN(NIL);
          END LOOKUP;

        /*************************************************
        *    PROCEDURE WHICH SCANS THE INPUT BUFFER      *
        *    AREA AND RETURNS VALID CHARACTER STRINGS    *
        *    WITHOUT BLANKS TO THE CALLING POINT.  THE   *
        *    VALID CHARACTER STRINGS ARE SINGLE SPECIAL  *
        *    (EXCEPT  '-') AND ALFANUMERIC STRINGS       *
        *    INCLUDING THE '-' CHARACTER.                *
        *************************************************/

       DCL SCAN RETURNS(CHAR(80)VAR);
       SCAN: PROC CHAR(80)VAR;
          DCL ((BP INIT(72),B)FIXED BIN,(T CHAR(1),
             BUFF CHAR(72)))STATIC;
          B = 0;
             DO WHILE(B=0);
             IF BP >= 72 THEN DO;
                BP = 0; CALL INPUT(BUFF); END;
             BP = BP+1;
             IF SUBSTR(BUFF,BP,1)¬=' ' THEN B=BP; END;
          T=SUBSTR(BUFF,B,1);
          IF T < 'A' & T ¬= '-' THEN RETURN(T);
             DO WHILE(BP<72 & T ¬=' ' & (T ='-' | T >='A'));
             BP = BP+1; T=SUBSTR(BUFF,BP,1); END;
          BP=BP-1;
          RETURN(SUBSTR(BUFF,B,BP-B+1));
          END SCAN;

        /*************************************************
        *    PROCEDURE WHICH CONSTRUCTS A CONNECTIVE     *
        *   CELL (A CELL WITH A 00 ATOMB FIELD)          *
        *    WITH THE INPUT POINTERS AND THE             *
        *    CURRENT LEVEL.                              *
        *************************************************/

       DCL CONS ENTRY(PTR,PTR) RETURNS(PTR);
```

```
CONS: PROC(A,B) PTR;
    DCL (A,B)PTR;
    ALLOCATE CONCELL SET(P);
    ATOMB='00'B;
    LEVEL = LEV;
    CARF=A;
    CDRF=B;
    RETURN(P);
    END CONS;

  /***********************************************************
  *      PROCEDURE WHICH RETURNS THE POINTER                 *
  *      CONTAINED IN THE CARF FIELD OF THE                  *
  *      SPECIFIED CELL.                                     *
  ***********************************************************/

DCL CAR ENTRY(PTR) RETURNS(PTR);
CAR: PROC(A) PTR;
    DCL A PTR;
    IF A=NIL THEN RETURN(NIL);
    IF A->ATOMB THEN RETURN(NIL);
    RETURN(A->CARF);
    END CAR;

  /***********************************************************
  *      PROCEDURE WHICH RETURNS THE POINTER                 *
  *      CONTAINED IN THE CDRF FIELD OF THE                  *
  *      SPECIFIED CELL.                                     *
  ***********************************************************/

DCL CDR ENTRY(PTR) RETURNS(PTR);
CDR: PROC(A) PTR;
    DCL A PTR;
    IF A=NIL THEN RETURN(NIL);
    RETURN(A->CDRF);
    END CDR;

  /***********************************************************
  *      PROCEDURE WHICH RETURNS THE POINTER                 *
  *      CONTAINED IN THE CARF FIELD OF THE CELL             *
  *      POINTED TO BY THE CARF FIELD OF THE                 *
  *      SPECIFIED CELL.                                     *
  ***********************************************************/

 DCL CAAR ENTRY(PTR) RETURNS(PTR);
 CAAR: PROC(A) PTR;
     DCL A PTR;
     IF A = NIL THEN RETURN(NIL);
     A = A->CARF;
     RETURN(A->CARF);
     END CAAR;

  /***********************************************************
  *      PROCEDURE WHICH WILL PRINT OUT ALL OF THE           *
  *      INDIVIDUAL CHARACTERS ON THE LIST SPECIFIED.        *
  ***********************************************************/

DCL PRINT ENTRY(PTR);
PRINT: PROC(A) RECURSIVE;
    DCL A PTR;
    IF A¬=NIL THEN
        IF A->ATOMB THEN CALL OUTPUT(SYMBOL(A));
        ELSE DO;
            CALL PRINT((A->CARF));
            CALL PRINT((A->CDRF)); END;
    END PRINT;

 /***********************************************************
 *      PROCEDURE WHICH RETURNS THE CHARACTERS              *
 *      CONTAINED IN THE PNAME FIELD OF THE                 *
 *      ATOMCELL SPECIFIED                                  *
 ***********************************************************/
```

```
DCL SYMBOL ENTRY(PTR) RETURNS(CHAR(80)VAR);
SYMBOL: PROC(X) CHAR(80)VAR;
   DCL (X,Y)PTR;
   IF X=NIL THEN RETURN('NIL');
   Y=X->CARF;
   RETURN(Y->PNAME);
   END SYMBOL;

  /************************************************************
  *     PROCEDURE WHICH IS USED TO CALL THE                    *
  *     PROCEDURE TO FORM THE LIST STRUCTURE.  IT              *
  *     IS ALSO USED TO ESTABLISH A POINTER TO THE             *
  *     MULTIPLE DEFINITIONS LIST DURING THE                   *
  *     THE INITIALIZATION PHASE.                              *
  ************************************************************/

DCL READSS ENTRY RETURNS(PTR);
READSS: PROC PTR RECURSIVE;
   DCL ((Q,P)PTR,CK FIXED BIN INIT(0))STATIC;
   ERR = '0'B; CK = CK+1;
   Q = S_EXPR('0'B);
   IF ERR THEN CALL OUTPUT('ILL FORMED EXPRESSION');
   IF CK = 1 THEN DO;
      PP,L3 = Q;
      IF TRI THEN DO;
         P = CAAR((L3));
         DISPLAY('FIRST ITEM ON MULTIPLE DEFINITION '
            ||'LIST  '||P->PNAME); END;
      CK = CK+1; END;
   ERR = '0'B; CK = CK-1;
   RETURN(Q);
   END READSS;

  /************************************************************
  *     PROCEDURE WHICH IS USED TO CALL THE LEVEL              *
  *     ZERO PROCEDURE WHICH BUILDS THE LIST                   *
  *     STRUCTURE AND CONTROLS THE DURATION OF                 *
  *     THE BUILDING.                                          *
  ************************************************************/

DCL READS ENTRY RETURNS(PTR);
READS: PROC PTR RECURSIVE;
   DCL Q PTR STATIC;
   ERR = '0'B;
   Q = S_EXPR('1'B);
   IF ERR THEN CALL OUTPUT('ILL FORMED EXPRESSION');
      DO WHILE(SCAN¬='$'); END;
   ERR = '0'B;
   RETURN(Q);
   END READS;

  /************************************************************
  *     PROCEDURE WHICH BUILDS THE 'LISP LIKE'                 *
  *     STRUCTURE OF THE LISTS USED BY THE PROGRAM.            *
  ************************************************************/

DCL S_EXPR ENTRY(BIT(1)) RETURNS(PTR);
S_EXPR: PROC(DOT) PTR RECURSIVE;
   DCL DOT BIT(1),(Q,R)PTR,(T,SYM)CHAR(80)VAR STATIC;
   DCL RIGHTSIDE RETURNS(PTR);
   RIGHTSIDE: PROC PTR;
      DCL P PTR;
      IF DOT THEN RETURN(Q);
      R=S_EXPR('0'B);
      IF ERR THEN RETURN(NIL);
      LEV = Q->LEVEL;
      P = Q->CARF;
      IF TRC THEN DISPLAY('LEFT CONNECTIVE SYMBOL  '
         ||P->PNAME||'  AT LEVEL'||LEV);
      IF R ¬= NIL THEN
         IF R->LEVEL = LEV+1 THEN DO;
```

```
                Q->CDRF = R;
                R = S_EXPR('0'B); END;
            RETURN(CONS(Q,P));
            END RIGHTSIDE;
        T=SCAN;
        IF T='(' THEN DO;
            Q=S_EXPR('0'B);
            IF ERR THEN RETURN(NIL);
            RETURN(RIGHTSIDE); END;
        IF T = '<' THEN DO;
            LEV = LEV +1;
            RETURN(READSS); END;
        IF T ='>' THEN DO;
            IF DOT THEN ERR ='1'B;
            ELSE RETURN(NIL); END;
        IF T=')' THEN DO;
            IF DOT THEN ERR = '1'B;
            ELSE RETURN(NIL); END;
        IF T = '/' THEN DO;
            SYM = SCAN;
            L3 = PP;
            ENC ='1'B;
            Q = CAR(LOOKLIST(L3,SYM));
            ENC ='0'B;
            R = S_EXPR('0'B);
            RETURN(CONS(Q,R)); END;
        ELSE DO;
            IF T = 'NIL' THEN Q = NIL;
            ELSE Q = STRING(T);
            RETURN(RIGHTSIDE); END;
        RETURN(NIL);
        END S_EXPR;

      /**************************************************
      *    PROCEDURE WHICH IS USED TO CONVERT THE ITEMS *
      *    IN THE LIST STRUCTURE INTO A FORMAT CAPABLE  *
      *    OF BEING PRINTED OUT.                        *
      **************************************************/

    DCL BUILD ENTRY(PTR,BIT(1)) RETURNS(CHAR(1000)VAR);
    BUILD: PROC(P,B) CHAR(1000)VAR RECURSIVE;
        DCL P PTR,B BIT(1);
        IF P=NIL THEN RETURN('NIL');
        IF ATOM(P) THEN DO;
            IF P ¬=NIL & P->CDRF ¬=NIL THEN RETURN(SYMBOL(P)
                ||'<'||BUILD(P->CDRF,'1'B)||'>');
            ELSE RETURN(SYMBOL(P)); END;
        IF ATOM(CDR(P)) THEN DO;
            IF CDR(P) = NIL THEN
                IF B THEN RETURN(BUILD(CAR(P),'0'B));
                ELSE RETURN('('||BUILD(CAR(P),'1'B)||')'); END;
        IF B THEN RETURN(BUILD(CAR(P),'0'B)||' '||
            BUILD(CDR(P),'1'B));
        RETURN('('||BUILD(P,'1'B)||')');
        END BUILD;

      /**************************************************
      *    PROCEDURE WHICH IS USED TO CALL THE          *
      *    PROCEDURE WHICH IS USED TO PRINT OUT THE     *
      *    ITEMS STRUCTURED BY THE BUILD PROCEDURE.     *
      **************************************************/

    DCL PRINTS ENTRY(PTR);
    PRINTS: PROC(A);
        DCL A PTR,B CHAR(1000)VAR,(L,I)STATIC FIXED BIN;
        DISPLAY('FORMATING LIST FOR PRINTING');
        IF TRI THEN DISPLAY('DICTIONARY PRINTING TAKES '
            ||'A LONG TIME');
        B = BUILD(A,'0'B);
        L = LENGTH(B);
            DO I=1 TO L BY 60;
            CALL OUTPUT(SUBSTR(B,I,MIN(60,L-I+1))); END;
```

```
   IF FLD THEN CALL OUTPUT('$');
   END PRINTS;

  /**************************************************
  *     PROCEDURE WHICH IS USED TO SEARCH DOWN A      *
  *     SPECIFIED LIST TO FIND THE CHARACTER(S)       *
  *     SPECIFIED BY THE PARAMETER.  THE POINTER TO   *
  *     THE NEXT ITEM 'DOWN' THE LIST IS RETURNED.    *
  **************************************************/

DCL LOOKLIST ENTRY(PTR,CHAR(*)VAR) RETURNS(PTR);
LOOKLIST: PROC(P,C) PTR;
   DCL (P,Q,R)PTR,C CHAR(*)VAR;
     DO WHILE (P¬=NIL);
     Q = P->CARF;
     R = CAAR((P));
     IF R->PNAME = C THEN RETURN(Q->CDRF);
     ELSE IF ¬ENC THEN
        IF Q->ATOMB = '11'B THEN
           IF L3 = NIL THEN DO;
              L3 = CONS(Q,NIL);
              L3->LEVEL = 1; END;
           ELSE DO;
              L3 = CONS(Q,L3);
              L3->LEVEL = 1; END;
        P = P->CDRF; BP = P; END;
   RETURN(NIL);
   END LOOKLIST;

  /**************************************************
  *     PROCEDURE WHICH IS USED TO ATTACH TO ALL      *
  *     ITEMS ON THE 'L2' LIST (THE USER DEFINED      *
  *     LIST) ANY ATTRIBUTES OF THAT ITEM FOUND ON    *
  *     THE 'L1' (DICTIONARY) LIST.                   *
  **************************************************/

DCL ATTRB ENTRY;
ATTRB: PROC;
   DCL (P,Q,R,S,T)PTR,C CHAR(20)VAR,L FIXED BIN,
     C1 CHAR(1);
   R = L2; L3 = NIL;
     DO WHILE(R ¬=NIL);
     P = L1;
     S = R->CARF;
     T=CAAR((R));
     C = T->PNAME;
     L = T->SIZE;
        DO I = 1 TO L;
        C1 = SUBSTR(C,I,1);
        Q = LOOKLIST(P,C1);
        IF Q = NIL THEN DO;
           DISPLAY('ATTRIBUTE NOT FOUND FOR ** '||C);
           GO TO LB1; END;
        ELSE P = Q; END;
     Q = LOOKLIST(P,'*');
     Q = DELETE(Q);
     IF Q = NIL THEN S->CDRF = L3;
     ELSE DO;
        IF L3 ¬= NIL THEN DO;
           S->CDRF = L3;
              DO WHILE(L3->CDRF¬=NIL);
              L3 = L3->CDRF; END;
           L3->CDRF = Q; END;
        ELSE S->CDRF = Q; END;
     DISPLAY('ATTRIBUTES FOUND FOR  '||C);
     LB1: R = R->CDRF; L3 = NIL; END;
  END ATTRB;

  /**************************************************
  *     PROCEDURE WHICH IS USED TO DELETE FROM THE    *
  *     LIST OF ATTRIBUTES THOSE ATTRIBUTES WHICH     *
  *     WERE PICKED UP DURING THE SEARCH WHICH DO     *
  *     NOT APPLY TO THE SPECIFIC ITEM.                          *
  ***********************************************************/

DCL DELETE ENTRY(PTR) RETURNS(PTR):
DELETE: PROC(Q)PTR;
   DCL (P,Q,R,S,T,BQ)PTR,C CHAR(20)VAR:
   BQ = Q:
      DO WHILE(Q ¬= NIL):
      S = CAAR((Q)):
      IF SUBSTR(S->PNAME,1,1) = '-' THEN DO:
         BQ = Q->CDRF:
         C = SUBSTR(S->PNAME,2):
         P,R = L3:
            DO WHILE(R ¬= NIL):
            T = CAAR((R)):
            IF T->PNAME = C THEN DO:
               T = R->CDRF:
               IF P = R THEN L3 = T:
               ELSE P->CDRF = T:
               GO TO LB2: END:
            ELSE R = R->CDRF: END:
    LB2: Q = BQ: END:
      ELSE RETURN(BQ): END:
   RETURN(BQ):
   END DELETE:

  /**********************************************************
  *     PROCEDURE FOR THE ON LINE ENTRY OF ITEMS             *
  *     INTO THE DICTIONARY LIST STRUCTURE.                  *
  **********************************************************/

DCL DEFINE ENTRY:
DEFINE: PROC RECURSIVE:
   DCL C1 CHAR(1):
   DISPLAY('ENTER WORD FOR DICTIONARY ENTRY')
      REPLY(IPT):
   L = LENGTH(IPT):
   BP,L2 = L1:
      DO I = 1 TO L:
      C1 = SUBSTR(IPT,I,1):
      L3 = LOOKLIST(L2,C1):
      IF I =1 THEN PP = L3:
      IF L3 = NIL THEN DO:
         IF BP = L2 THEN IPT = SUBSTR(IPT,1,I):
         ELSE IPT = SUBSTR(IPT,1,I-1):
         DISPLAY('LETTERS ALREADY IN DICTIONARY '||IPT):
         CALL PRINTS(PP):
         DISPLAY('ENTER REMAINDER OF WORD AND '
            ||'ITS ATTRIBUTES IN DICTIONARY FORM'):
         BP->CDRF = READS: END:
      ELSE BP,L2 = L3: END:
   CALL PRINTS(PP):
   DISPLAY('DICTIONARY COMPLETE = 0, ELSE 1')
      REPLY(IPT):
   FLD = IPT: IPT = '':
   IF FLD THEN CALL DEFINE:
   END DEFINE:

  /**********************************************************
  *     PROCEDURE WHICH IS USED TO CALL THE HASHX            *
  *     SUBROUTINE TO RETRIEVE THE NUMERIC CODE              *
  *     WORDS ASSOCIATED WITH THE TOPIC NAME.                *
  **********************************************************/

 TABLE: PROC(C):
    DCL C CHAR(20), TBLCODE FIXED BIN INIT(0):
    CALL HASHX(C,CODE,5,1,1,TBLCODE):
    IF TBLCODE = 0 THEN DISPLAY('  TOPIC NAME AND '
       ||'NUMERIC CODE WORD ** '||C):
    ELSE DISPLAY('  ** TOPIC NOT IN TABLE **  '||C):
    END TABLE:
```

```
  /*************************************************
  *    DECLARATION OF THE GLOBAL VARIABLES USED     *
  *    AND DISPLAY OF COMMAND REQUESTS TO THE       *
  *    OPERATOR.                                    *
  *************************************************/

DCL (L1,L2,L3,BP,PP)PTR,STNAME CHAR(40),IPT CHAR(80)VAR,
   (TRC,ENC,TRI,ERR,FLD,TBL,DCT)BIT(1),
   (I,L,LEV,TBLSIZE)FIXED BIN;
TRC,TRI,ENC,ERR,FLD,TBL,DCT='0'B; LEV=0; TBLSIZE=11;

  /*************************************************
  *    INITIALIZATION OF THE DICTIONARY LIST 'L1'.  *
  *************************************************/

IF PARMS ¬= '' THEN DISPLAY('ENTER OPTIONAL PARAMETERS')
   REPLY(IPT);
IF IPT ¬= '' THEN GET STRING(IPT) DATA(TRC,TRI,FLD,
   TBL DCT); IPT = '';
CALL INIT;

  /*************************************************
  ***************************************************
  **   ROUTINE FOR THE OPERATION OF THE PROGRAM.   **
  ***************************************************
  *************************************************/

   DO WHILE('1'B);
   DISPLAY(' ');
   DISPLAY('ENTER NAME AND SMC NUMBER') REPLY(STNAME);
   DISPLAY(' ');
   IF TRI THEN DISPLAY('ENTER OPTIONAL PARAMETERS,' ||
      'IF NONE THEN RETURN') REPLY(IPT);
   IF IPT ¬= '' THEN GET STRING(IPT) DATA(TRC,ENC,
      TRI,FLD,TBL,DCT); IPT = '';
   IF DCT THEN CALL DEFINE;
   ELSE DO;
      L2=READS;
      CALL PRINT(L2);
      IF ATOM(L2) THEN IF SYMBOL(L2)='STOP' THEN EXIT;
      DISPLAY(' ');
      DISPLAY('      ******************************');
      DISPLAY('          '||STNAME);
      DISPLAY('      ******************************');
      DISPLAY(' ');
      CALL ATTRB;
      DISPLAY(' ');
      CALL PRINTS(L2); END;
   IF TBL THEN CALL TABLE(' **  TOPIC NAME  ** ');
   DISPLAY(' ');
   DISPLAY('** AT THIS POINT YOUR REQUEST WOULD'||
      ' BE PASSED TO THE ');
   DISPLAY('   TRANSLATION AND TABLE SEARCH '||
      'ROUTINES. **');
   DISPLAY(' '); END;
END LIB_SER;
```

```
      /**************************************************
      **************************************************
      **                                              **
      **              TABLE PROGRAM                   **
      **                                              **
      **************************************************
      **************************************************/

TABLE: PROC(PARMS) OPTIONS(MAIN);
  DCL CODES FILE STREAM ENVIRONMENT (F(80));
  DCL (TRC,SWT)BIT(1),(CODE(11),ALFA)CHAR(20),
     (I,DATASIZE,NRKEY,TBLSIZE,TYPOP,TBLCODE)FIXED BIN,
     (PARMS,IPT)CHAR(2)VAR;

    /**************************************************
    *    DECLARATION OF A FILE CONTROL BLOCK FOR      *
    *    USE IN THE READIN OF THE EXTERNAL FILE.      *
    **************************************************/

  DCL 1 FCB STATIC,
      2 COMMAND CHAR(8),
      2 FILENAME CHAR(8) INIT('CODES'),
      2 FILETYPE CHAR(8) INIT('DATA'),
      2 CARDNUM FIXED BIN,
      2 STATUS FIXED BIN,
      2 CARD_BUFFER CHAR(80);

    /**************************************************
    *              PROGRAM OPERATION                  *
    **************************************************/

  IF PARMS ¬= '' THEN TRC = '1'B;
  ELSE TRC = '0'B;
  TBLSIZE = 11; TBLCODE = 0;
  DATASIZE = 5; NRKEY = 1;

    /**************************************************
    *    ROUTINE TO READ THE CODES FILE INTO THE      *
    *    ARRAY FOR  USE DURING THE PROGRAM.           *
    **************************************************/

  DISPLAY('INITIALIZATION OF THE CODE ARRAY');
  COMMAND = 'RDBUF';
     DO I = 1 TO TBLSIZE;
     CARDNUM = I;
     CALL IHEFILE(FCB);
     CODE(I) = CARD_BUFFER;
     IF TRC THEN DISPLAY('CODE('||I||')   '||CODE(I)); END;
  COMMAND = 'FINIS';
  CALL IHEFILE(FCB);
  DISPLAY('COMPLETION OF INITIALIZATION');

    /**************************************************
    *    ROUTINE WHICH ALLOWS THE CONTINUAL STORE,    *
    *    DELETION, OR RETRIEVAL OF ITEMS FROM THE     *
    *    ARRAY UNTIL THE OPERATOR IS FINISHED.        *
    **************************************************/

  SWT ='1'B;
     DO WHILE(SWT);
     DISPLAY('ENTER TYPE OF OPERATION: STORE=0,RETRIEVE=1'
        ||',DELETE=-1') REPLY(IPT);
     TYPOP = IPT;
     DISPLAY('ENTER ITEM REQUIRED') REPLY(ALFA);
     IF TRC THEN DISPLAY('CALL HASHX');

    /**************************************************
    *     THE FOLLOWING ARE THE PARAMETERS REQUIRED:  *
```

NOT REPRODUCIBLE

```
*     ALFA = ITEM UNDER CONSIDERATION                         *
*     CODE = THE TABLE ARRAY AREA                             *
*     DATASIZE = NUMBER OF COMPUTER WORDS IN                  *
*     THE ITEM BEING CONSIDERED                               *
*     NRKEY = NUMBER OF ITEMS, NORMALLY ONE                   *
*     TYPOP = TYPE OF OPERATION                               *
*     TBLSIZE = THE TABLE SIZE (PRIME NUMBER)                 *
*     TBLCODE = THE TABLE CONFIGURATION CODE (0)              *
*********************************************************/

 CALL HASHX(ALFA,CODE,DATASIZE,NRKEY,TYPOP,TBLSIZE,
    TBLCODE);

/*********************************************************
*     THE FOLLOWING ARE THE TABLE CODES WHICH               *
*     RESULT FROM THE CALL TO HASHX:                        *
*           0 = OPERATION SUCCESFULLY COMPLETED             *
*           1 = TABLE IS FULL                               *
*           2 = ITEM IS NOT IN THE TABLE                    *
*********************************************************/

 DISPLAY('TABLE CODE '||TBLCODE);
 IF TYPOP = 1 THEN DISPLAY('RETRIEVED ITEM AND CODE:   '
    ||ALFA);
 DISPLAY('IF FINISHED ENTER 0, ELSE 1') REPLY(IPT);
 SWT = IPT; END;

/*********************************************************
*     ROUTINE USED TO WRITE THE ARRAY BACK INTO             *
*     THE CODES FILE.                                       *
*********************************************************/

COMMAND = 'ERASE';
CALL IHEFILE(FCB);
COMMAND = 'WRBUF';
   DO I = 1 TO TBLSIZE;
   CARDNUM = I;
   IF TRC THEN DISPLAY('CODE('||I||')   '||CODE(I));
   CARD_BUFFER = CODE(I);
   CALL IHEFILE(FCB); END;
END TABLE;
```

NOT REPRODUCIBLE

```
*
*
*     /**************************************************
*     **************************************************
*     **                                              **
*     **              HASHX SUBROUTINE                **
*     **                                              **
*     **************************************************
*     **************************************************/
*
*
         MACRO
&N1      HAKEY
*        MACRO USED TO CONSTRUCT THE HASH CODE DISPLACEMENT
&N1      LM    4,5,TEMP+24  LOAD THE COLLISION CONSTANTS
         LM    6,7,0(11)    ESTABLISH HASH CODE INDEX
         XR    6,7          REDUCE THE KEY TO ONE WORD
         SRDA  6,32
         D     6,TEMP+12    HASH CODE INDEX
         LR    7,6
         MR    6,2          HASH TABLE DISPLACEMENT
         C     7,=F'0'
         BNL   *+6
         LCR   7,7          LOAD COMPLEMENT IF NEGATIVE
         LR    12,7
         AR    12,15        ESTABLISH HASH TABLE ENTRY POINT
         MEND
*
         MACRO
&N2      KEYCK &BP1
*        MACRO USED TO COMPARE THE INPUT WITH THE STORED ITEM
&N2      LM    8,10,0(11)   LOAD THE FIRST THREE WORDS OF DATA
         CL    8,0(12)      CHECK THE FIRST WORD FOR MATCH
         BNE   &BP1         BRANCH IF NOT SAME
         CL    9,4(12)      CHECK THE SECOND WORD
         BNE   &BP1         BRANCH IF NOT SAME
         CL    10,8(12)     COMPARE THE THIRD WORD FOR MATCH
         BNE   &BP1         BRANCH IF NOT SAME
         MEND
*
         MACRO
&N3      HCOLL &BP2,&BP3
*        MACRO USED TO COMPUTE THE COLLISION DISPLACEMENT
&N3      AR    4,2          ADD DATA SIZE TO LOWER COLL CONST
         CR    4,5          CHECK TO SEE IF WITHIN TABLE AREA
         BNL   &BP3         IF NOT GO TO TABLE FULL ROUTINE
         BXLE  7,4,*+6
         SR    7,5
         LR    12,7
         AR    12,15        RECOMPUTE NEW HASH TABLE ENTRY
         B     &BP2
         MEND
*
         MACRO
&N4      MVCHA &R1,&R2,&BP1
*        MACRO USED TO MOVE THE DATA FROM INPUT TO HASH TABLE
*        OR FROM HASH TABLE TO OUTPUT
&N4      AR    2,1          REDUCE NUMBER OF CHAR BY ONE
         EX    2,*+8        MODIFY THE NUMBER OF CHAR MOVED
         B     *+10
         MVC   0(1,&R1.),0(&R2.)
         SR    2,1          RESTORE THE NUMBER OF CHARACTERS
         AR    11,2
         BCT   3,&BP1       CHECK TO SEE IF ALL DATA PROCESSED
         B     EXIT         IF FINISHED BRANCH TO EXIT ROUTINE
         MEND
*
         MACRO
&N5      TBLFUL &NR
&N5      L     1,TEMP+16
         L     2,=F'&NR.'
         ST    2,0(1)
```

NOT REPRODUCIBLE

```
         B      EXIT
         MEND
*
*
*        AFTER INITIALIZATION THE FOLLOWING ARE THE
*        REGISTER ASSIGNMENTS.
*        R-0 CONTAINS A CONSTANT BLANK FOR COMPARISONS
*        R-1 CONTAINS A CONSTANT MINUS ONE
*        R-2 CONTAINS THE DATA LENGTH IN BYTES
*        R-3 CONTAINS THE NUMBER OF ITEMS TO BE PROCESSED
*        R-4 CONTAINS A COLLISION CONSTANT ((P#-1)/2*R2)
*        R-5 CONTAINS A COLLISION CONSTANT  (P#*R2)
*        R-11 CONTAINS THE CURRENT DATA ADDRESS
*        R-12 CONTAINS THE HASH CODE DISPLACEMENT
*        R-13 CONTAINS THE PROGRAM BASE ADDRESS
*        R-14 CONTAINS THE DATA BASE ADDRESS
*        R-15 CONTAINS THE HASH TABLE BASE ADDRESS
*        R-6 TO R-10 ARE WORKING REGISTURES
*
HASHX    CSECT
         USING *,15
         B      12(0,15)       BRANCH AROUND IDENTIFIER
         DC     AL1(6)
         DC     CL6'HASHX'     IDENTIFIER
         STM    14,12,12(13)   SAVE REGISTURES
         LR     12,13
         CNOP   0,4
         BAL    13,*+76
         DROP   15
         USING  *,13
         DS     18F
         ST     13,8(12)
         ST     12,4(13)
*
*        BEGIN PROGRAM
*
         LM     14,15,0(1)     LOAD KEY ADDRESS AND HASH TABLE BASE
         LM     2,6,8(1)       LOAD PARAMETERS
         L      2,0(2)         LOAD KEY AND DATA SIZE
         L      3,0(3)         LOAD NUMBER OF KEYS OR DATA
         L      4,0(4)         LOAD TYPE OF OP(RET=1,ST=0,DEL=-1)
         L      5,0(5)         LOAD TABLE SIZE (PRIME NUMBER)
         CNOP   0,4
         B      *+36
TEMP     DS     8F             STORAGE SPACE FOR PARAMETERS
         STM    2,6,TEMP
         SLA    2,2            KEY/DATA SIZE TIMES FOUR
         ST     2,TEMP+20      TEMP STORE OF DATA SIZE IN BYTES
         LR     7,5            ESTABLISH COLLISION CONSTANTS TO #
         BCTR   5,0
         SRA    5,1            SUBTRACT ONE FROM TABLE SIZE
         MR     4,2            MULTIPLY TABLE SIZE BY DATA SIZE
         SLA    7,2            MULTIPLY TABLE SIZE BY FOUR
         MR     6,2            MULTIPLY 4-TABLE SIZE BY DATA SIZE
         ST     5,TEMP+24      #-- TEMP STORE OF CONSTANTS
         ST     7,TEMP+28
         LR     11,14
         L      1,=F'-1'       ESTABLISH CONSTANT MINUS ONE
         L      0,=C'    '     ESTABLISH CONSTANT BLANK
         L      6,=F'0'        ESTABLISH ZERO CONSTANT
         C      6,TEMP+8       CHECK TO SEE IF STORE OPERATION
         BE     LOP1           BRANCH TO STORE ROUTINE
         C      1,TEMP+8       CHECK TO SEE IF DELET OPERATION
         BE     DELT           BRANCH TO DELETION ROUTINE
         B      RTRV           BRANCH TO RETRIEVE ROUTINE
*
*        STORAGE ROUTINE
*
LOP1     HAKEY
LOP2     C      0,0(12)        CHECK FOR EMPTY SLOT
         BE     *+16           IF EMPTY BRANCH TO CHARACTER MOVE
         C      1,0(12)        CHECK TO SEE IF ITEM DELETED
```

```
          BE     *+8            IF DELETED BRANCH TO MOVE ROUTINE
          B      FSLT           BRANCH TO FULL SLOT ROUTINE
          MVCHA  12,11,LOP1
     FSLT KEYCK  CLS2
          MVCHA  12,11,LOP1
     CLS2 HCOLL  LOP2,TFL1
     *
     *    RETRIEVE ROUTINE
     *
     RTRV HAKEY
     LOP3 KEYCK  CLS1
          MVCHA  11,12,RTRV
     CLS1 HCOLL  LOP3,TFL2
     *
     *    DELETION ROUTINE
     *
     DELT HAKEY
     LOP4 KEYCK  CLS3
          ST     1,0(12)
          ST     1,4(12)
          BCT    3,DELT
     CLS3 HCOLL  LOP4,TFL2
     *
     *    TABLE FULL ROUTINE
     *
     TFL1 TBLFUL 1
     TFL2 TBLFUL 2
     *
     *    RETURN ROUTINE
     *
     EXIT L 13,4(13)
          LM     14,12,12(13)       RESTORE REGISTURES
          MVI    12(13),X'FF'       SET RETURN INDICATION
          LA     15,0(0,0)          LOAD RETURN CODE
          BR     14                 RETURN
          END
```

```
/***********************************************
************************************************
**                                            **
**              DICTIONARY FILE               **
**                                            **
************************************************
***********************************************/

(1<MSPLR<S<*<PLUR>>> MESPLR<E<S<*<PLUR>>>> MLY<L<Y<*<-ADJP
ADVBP>>>> MION<I<O<N<*<-VERBP NOUNP>>>>> MING<I<N<G<*<PRSPT
>>>>> MAL<A<L<*<-NOUNP ADJP>>>> MED<E<D<*<PAST>>>> MEST<E<S
<T<*<SUPRL>>>>> MER<E<R<*<COMPR>>>> MIC<I<C<*<-NOUNP ADJP>>
>>MMENT<M<E<N<T<*<-VERBP NOUNP>>>>>> MABLE<A<B<L<E<*<-VERBP
ADJP>>>>>> MICS<I<C<*<-NOUNP ADJP> S<*>>>> MD<D<*<PAST>>>>

E    <L<E<C<T<R<NOUNP O<N</MICS> D<E<*>> L<Y<S<I<S<*>>>>>> I
     <C<I<T<Y<*>>> /MAL>>>>>>>
T    <O<*<PREPP ADVBP>> H<E<*<ADJP DEFART>> A<T<*<PPNOUNP
     ADJP>>>> E<L<E<M<E<T<R<Y<*<NOUNP>>>>>>>>>
A    <*<ADJP INDART> N<*<ADJP INDART>> L<L<*<ADJP>>> C<O<U<S
     <NOUNP T</MICS>>>> O<U<I<P<VERBP E<* /MD> /MING>>>>> B<
     O<U<T<*<PREPP>>>>> M<*<VERBP>>>
O    <B<T<A<I<N<VERBP * /MING>>>>> PREPP N<*> F<*> U<T<*<
     -PREPP ADVBP>>>>
I    <*<PRNOUN> N<*<PREPP> F<NOUNP O<* R<M<*<-NOUNP VERBP> A
     <T</MION>>>>>> T<E<R<E<S<T<VERBP * S<*> /MED>>>>>>> S<*
     <VERBP>>>
N    <A<T<U<R<E<*<NOUNP>> /MAL>>>>>
S    <O<M<E<*<ADVBP>>>> E<N<D<VERBP * /MING>>>>
R    <E<F<E<R<VERBP * R</MING /MED>>>> P<O<R<T<NOUNP *
     /MSPLR>>>>>>
H    <A<VERBP S<*<PR3SG>> D<*<PAST>> V<E<*<PRN3SG>> /MING>>
     O<L<D<I<N<G<NOUNP * /MSPLR>>>>>>>
L    <A<N<G<U<A<G<E<NOUNP * /MSPLR>>>>>>> I<K<E<*<VERBP>>>
     NOUNP S<T<I<N<G<* /MSPLR>>>>> N<G<U<I<S<T</MICS>>>>>>>>
D    <O<VERBP * /MING> E<S<T<R<O<Y<F<R<NOUNP * /MSPLR>>>>>>>
     >>     U
C    <A<L<C<U<L<U<S<*<NOUNP>>>>>>>> O<U<P<L<E<R<NOUNP *
     /MSPLR>>>>> M<P<U<T<E<R<NOUNP * /MSPLR>>>>>>>>
F    <I<N<D<VERBP * /MING>>>>
M    <E<*<PRNOUNP>>>
W    <A<N<T<*<VERBP>>>> O<U<L<D<*<VERBP>>>>> I<T<H<*<PREPP>>
     >> H<I<C<H<*<ADJP>>>> A<T<*<INTER ADVBP>>>>>
Y    <O<U<PRNOUNP * R<*<POSSP>>>>>
G    <I<V<VERBP E<*> /MING>> A<V<E<VERBP PAST>>>>
P    <L<E<A<S<E<*<VERBP>>>>>>>
B    <I<B<L<I<O<G<R<A<P<H<NOUNP Y<*> I<C</MAL>>>>>>>>>>>>>
     V  K  X  J  O  Z
.    <*<PUNTP>>
,    <*<PUNTC>>
?    <*<PUNTI>>)$
```

LIST OF REFERENCES

1. University of Pennsylvania, REAL ENGLISH: A Translator to Enable Natural Language Man-Machine Conversation, by. H. Cautin, May 1969.

2. H. Cautin, M. Rubinoff, S. Bergman, and F. Rapp, "EASY ENGLISH, A Language for Information Retrieval Through a Remote Typewriter Console," Communications of the ACM, v. 11, pp. 693-696 October 1968.

3. J. A. Craing, S. C. Berzner, H. C. Caryney, and C. R. Longyer, "DEACON: Direct English Access and Control," Proceedings - Fall Joint Computer Conference, pp. 365-380, 1966.

4. C. H. Kellogg, 26 May 1967, CONVERSE - A System for the On-Line Description and Retrieval of Structured Data Using a Natural Language, System Development Corporation Report SP-2635.

5. C. H. Kellogg, "On-Line Translation of Natural Language Questions Into Artificial Language Queries," Information Storage and Retrieval, v. 4, pp. 287-307, 1968.

6. C. H. Kellogg, "A Natural Language Compiler for On-Line Data Management," Proceedings - Fall Joint Computer Conference, pp. 473-492, 1968.

7. International Business Machines Corporation Form C28-6571-4, IBM System/360 Operating System PL/I Language Specifications, 1965.

8. International Business Machines Corporation Form C28-6594-3, IBM System/360 Operating System PL/I (F) Programmers Guide, 1967.

9. International Business Machines Corporation Form C28-8201-1, IBM System/360 PL/I Reference Manual, 1968.

10. International Business Machines Corporation Form A22-6821-7, IBM System/360 Principles of Operation, 1968.

11. International Business Machines Corporation Form C28-6514-5, IBM System/360 Operating System Assembler Language, 1967.

12. C. Weissman, LISP 1.5 Primer, Dickenson Publishing Company, 1968.

13. N. Abramson, Information Theory and Coding, McGraw-Hill Book Company, 1963.

14. W. D. Maurer, "An Improved Hash Code for Scatter Storage," Communications of the ACM, v. 11, pp. 35-38, January 1968.

15. R. Morris, "Scatter Storage Techniques," Communications of the ACM, v. 11, pp. 38-44, January 1968.

Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY *(Corporate author)* Naval Postgraduate School Monterey, California 93940	2a. REPORT SECURITY CLASSIFICATION Unclassified
	2b. GROUP

3. REPORT TITLE
A Dictionary Structure for Use with an English Language Preprocessor to a Computerized Information Retrieval System

4. DESCRIPTIVE NOTES *(Type of report and, inclusive dates)*
Master's Thesis; June 1970

5. AUTHOR(S) *(First name, middle initial, last name)*
Lieutenant Commander Charles Thomas Schmidt

6. REPORT DATE June 1970	7a. TOTAL NO. OF PAGES 52	7b. NO. OF REFS 15
8a. CONTRACT OR GRANT NO.	9a. ORIGINATOR'S REPORT NUMBER(S)	
b. PROJECT NO.		
c.	9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*	
d.		

10. DISTRIBUTION STATEMENT
This document has been approved for public release and sale; its distribution is unlimited.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY Naval Postgraduate School Monterey, California 93940

13. ABSTRACT

This paper describes the formation of a dictionary list structure which can be used by an English language translator to enable natural language man-machine conversation directed towards the retrieval of information from a data bank. The hierarchical structure of the letters in a word and the placement of word attributes in this structure is discussed.

A computer program, which accepts as input an English language sentence and processes this sentence in conjunction with the dictionary list structure to obtain the attributes of the individual words, is described. The incorporation of this dictionary structure into a complete natural language information retrieval system is also discussed.

DD FORM 1 NOV 65 1473 (PAGE 1)
S/N 0101-807-6811

Security Classification

A-31408

Security Classification

14 KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Information Retrieval						
Natural Language						
Dictionary Structure						
List Structure						

DD FORM 1 NOV 65 1473 (BACK)
S/N 0101-807-6821

Security Classification

A-31409